Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

ПНИПУ

**Лабораторная работа по ООП**

**«№7»**

Выполнил:

студент группы РИС-23-2б

Ившин Максим Сергеевич

Проверила:

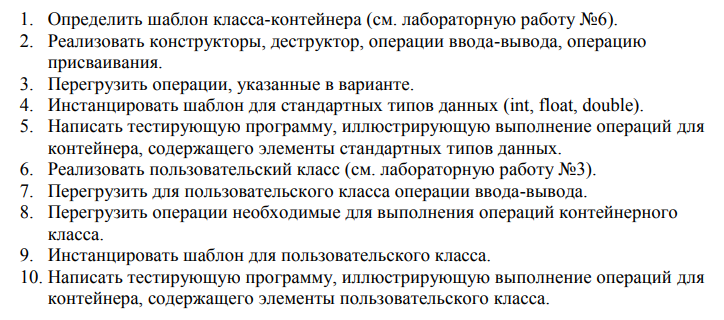
доцент кафедры ИТАС

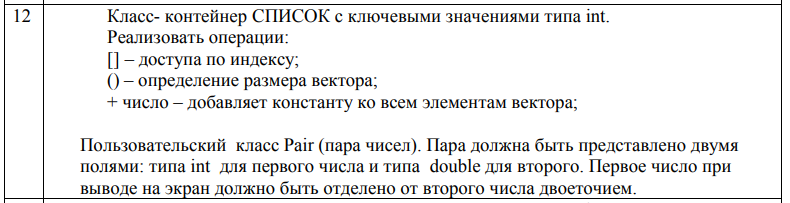
О.А. Полякова

2024 г.

**Разработка алгоритма**

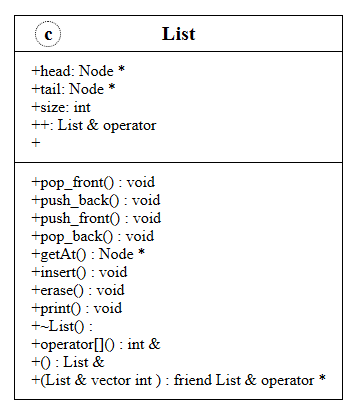
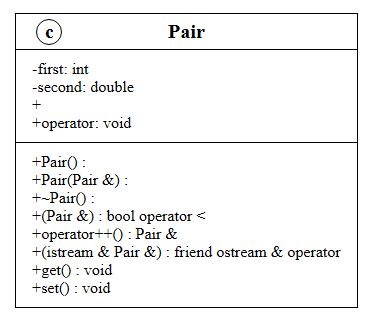
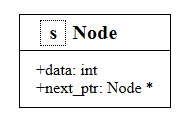
**Постановка задачи:**

****

****

**Анализ задачи:**

1. Создадим несколько заголовочных файлов и файлом с кодом для удобной работы.
2. Используем готовую реализацию класса-контейнера списка из лабораторной работы №6.
3. Также используем класс Pair из лабораторной работы №3.
4. На основе лабораторной работы №6 создадим шаблонный класс List, который позволит создавать список элементов произвольного типа данных.
5. Перегрузим операции [] доступа по индексу, () – определении размера и + – для добавления константы к элементам списка.



Код на С++:

main.cpp:

#include <iostream>

using namespace std;

#include "list.h"

#include "pair.h"

int main()

{

    // system("chcp 1251>null");

    List<int> list1(5);

    List<float> list2(5);

    List<double> list3(5);

    List<Pair> list4(3);

    List<int> list5(5);

    cout << "list1 int\n";

    cin >> list1;

    cout << list1;

    cout << "\nВывод элемента под номером 1: ";

    cout << list1[0] << "\n";

    cout << "\nlist2 float\n";

    cin >> list2;

    cout << list2;

    cout << "\nВывод элемента под номером 3: ";

    cout << list2[2] << "\n";

    cout << "\nlist3 double\n";

    cin >> list3;

    cout << list3 << "\n";

    cout << "\nlist4 Pair\n";

    cin >> list4;

    cout << list4;

    cout << "\nВывод элемента под номером 2:\n";

    cout << list4[1] << "\n";

    cout << "\nlist5 = list1\n";

    list5 = list1;

    cout << list5 << "\n";

    return 0;

}

pair.h:

#pragma once

#include <iostream>

using namespace std;

class Pair

{

private:

    int first;

    double second;

public:

    Pair();

    Pair(int, double);

    Pair(Pair &);

    ~Pair(){};

    bool operator<(Pair &);

    bool operator>(Pair &);

    Pair &operator++();

    Pair &operator++(int);

    friend ostream &operator<<(ostream &out, Pair i);

    friend istream &operator>>(istream &stream, Pair &pair);

    void operator=(Pair &);

    void get();

    void set(int, double);

};

Pair::Pair()

{

    this->first = 0;

    this->second = 0;

}

Pair::Pair(int fir, double sec)

{

    this->first = fir;

    this->second = sec;

}

Pair::Pair(Pair &copying)

{

    this->first = copying.first;

    this->second = copying.second;

}

bool Pair::operator<(Pair &temp)

{

    if ((this->first + this->second) < (temp.first + temp.second))

        return true;

    return false;

}

bool Pair::operator>(Pair &temp)

{

    if ((this->first + this->second) > (temp.first + temp.second))

        return true;

    return false;

}

Pair &Pair::operator++()

{

    ++this->first;

    return \*this;

}

Pair &Pair::operator++(int)

{

    ++this->second;

    return \*this;

}

void Pair::get()

{

    std::cout << "first = " << this->first << ", second = " << this->second;

}

void Pair::set(int fir, double sec)

{

    this->first = fir;

    this->second = sec;

}

std::ostream &operator<<(std::ostream &pairout, Pair temp)

{

    return (pairout << temp.first << ':' << temp.second);

}

istream &operator>>(istream &stream, Pair &pair)

{

    int first;

    double second;

    stream >> first >> second;

    pair.first = first;

    pair.second = second;

    return stream;

}

void Pair::operator=(Pair &other)

{

    this->first = other.first;

    this->second = other.second;

}

list.h:

#pragma once

#include <iostream>

template <typename T>

struct Node

{

    T data;

    Node<T>\* ptr\_to\_next\_node = nullptr;

};

template <typename T>

class List

{

private:

    int size;

    Node<T>\* head\_node;

    Node<T>\* tail\_node;

public:

    List(int);

    List(List<T>&);

    ~List();

    void push\_back(T);

    void pop\_front();

    int operator () ();

    T& operator[](int);

    List& operator = (List<T>&);

    List operator \*(List<T>&);

    friend std::ostream& operator << (std::ostream& stream, List<T>& list)

    {

        std::cout << "Вывод элементов: \n";

        Node<T>\* current\_node = list.head\_node;

        while (current\_node != nullptr)

        {

            stream << current\_node->data << ' ';

            current\_node = current\_node->ptr\_to\_next\_node;

        }

        return stream;

    }

    friend std::istream& operator >> (std::istream& stream, List<T>& list)

    {

        std::cout << "Введите элементы: ";

        Node<T>\* current\_node = list.head\_node;

        while (current\_node != nullptr)

        {

            stream >> current\_node->data;

            current\_node = current\_node->ptr\_to\_next\_node;

        }

        return stream;

    }

};

template <typename T>

List<T>::List(int size)

{

    if (size > 0)

    {

        Node<T>\* node = new Node<T>;

        this->head\_node = node;

        this->tail\_node = node;

        for (int i = 1; i < size; i++)

        {

            Node<T>\* New\_Node = new Node<T>;

            tail\_node->ptr\_to\_next\_node = New\_Node;

            tail\_node = New\_Node;

        }

        this->size = size;

    }

    else

    {

        this->head\_node = nullptr;

        this->tail\_node = nullptr;

    }

}

template <typename T>

List<T>::List(List& list)

{

    this->size = 0;

    Node<T>\* current\_node = list.head\_node;

    while (current\_node != nullptr)

    {

        push\_back(current\_node->data);

        current\_node = current\_node->ptr\_to\_next\_node;

    }

}

template <typename T>

List<T>::~List()

{

    while (head\_node != nullptr)

    {

        this->pop\_front();

    }

}

template <typename T>

void List<T>::push\_back(T data)

{

    Node<T>\* New\_Node = new Node<T>;

    New\_Node->data = data;

    if (this->head\_node == nullptr)

    {

        this->head\_node = New\_Node;

        this->tail\_node = New\_Node;

    }

    else

    {

        this->tail\_node->ptr\_to\_next\_node = New\_Node;

        this->tail\_node = New\_Node;

    }

    this->size++;

}

template <typename T>

void List<T>::pop\_front()

{

    if (this->tail\_node != nullptr)

    {

        Node<T>\* current\_node = this->head\_node;

        head\_node = current\_node->ptr\_to\_next\_node;

        this->size--;

    }

}

template <typename T>

int List<T>::operator () ()

{

    return this->size;

}

template <typename T>

T& List<T>::operator[](int index)

{

    Node<T>\* current\_node = this->head\_node;

    for (int i = 0; i != index; i++)

    {

        current\_node = current\_node->ptr\_to\_next\_node;

    }

    return current\_node->data;

}

template <typename T>

List<T>& List<T>::operator = (List<T>& list)

{

    if (this != &list)

    {

        while (head\_node != nullptr)

        {

            this->pop\_front();

        }

        size = 0;

        Node<T>\* current\_node = list.head\_node;

        while (current\_node != nullptr)

        {

            push\_back(current\_node->data);

            current\_node = current\_node->ptr\_to\_next\_node;

        }

    }

    return \*this;

}

template <typename T>

List<T> List<T>::operator \*(List<T>& list)

{

    int temp\_size;

    if (this->size > list.size)

    {

        temp\_size = list.size;

    }

    else

    {

        temp\_size = this->size;

    }

    List<T> temp\_list(temp\_size);

    for (int i = 0; i < temp\_size; i++)

    {

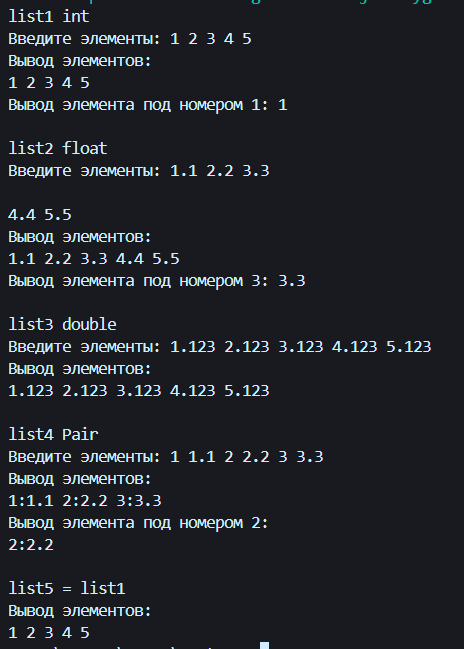
        temp\_list[i] = (\*this)[i] \* list[i];

    }

    return temp\_list;

}

Пример работы программы:



**GitHub:** *https://github.com/geroineee/PNRPU*